UNIT-II
PART - B

The Old Way and the New: The Principles of Conventional Software Engineering, Principles of Modern Software Management, Transitioning to an Iterative Process.

Introduction

- Over the past two decades there has been a significant re-engineering of the software development process.
- Many of the conventional management and technical practices have been replaced by new approaches that combine recurring themes of successful project experience with advances in software engineering technology.
- This transition was motivated by the insatiable demand for more software features produced more rapidly under more competitive pressure to reduce cost.
- In the commercial software industry, the combination of competitive pressures, profitability, diversity of customers, and rapidly changing technology caused many organizations to initiate new management approaches.

The Principles of Conventional Software Engineering

- There are many descriptions of engineering software "the old way."
- After years of software development experience, the software industry has learned many lessons and formulated many principles (nearly 201 by – Davis's). Of which Davis's top 30 principles are:

1. **Make quality #1**
   - Quality must be quantified and mechanisms put into place to motivate its achievement.

The Principles of Conventional Software Engineering

2. **High-quality software is possible**
   - In order to improve the quality of the product you need to involving the customer, select the prototyping, simplifying design, conducting inspections, and hiring the best people.
3. **Give products to customers early**
   - No matter how hard you try to learn user's needs during the requirements phase, the most effective way to determine real needs is to give users a product and let them play with it.
4. **Determine the problem before writing the requirements**
   - Whenever a problem is raised most engineers provide a solution. Before you try to solve a problem, be sure to explore all the alternatives and don't be blinded by the understandable solution.

The Principles of Conventional Software Engineering

5. **Evaluate design alternatives**
   - After the requirements are agreed upon, you must examine a variety of architectures and algorithms and choose the one which is not used.
6. **Use an appropriate process model**
   - For every project, there are so many prototypes (process models). So select the best one that is exactly suitable to our project.
7. **Use different languages for different phases**
   - Our industry's main aim is to provide simple solutions to complex problems. In order to accomplish this goal choose different languages for different modules/phases if required.
8. **Minimize intellectual distance**
   - You have to design the structure of a software is as close as possible to the real-world structure.

The Principles of Conventional Software Engineering

9. **Put techniques before tools**
   - An un disciplined software engineer with a tool becomes a dangerous, undisciplined software engineer.
10. **Get it right before you make it faster**
   - It is very easy to make a working program run faster than it is to make a fast program work. Don't worry about optimization during initial coding.
11. **Inspect the code**
   - Examine the detailed design and code is a much better way to find the errors than testing.
12. **Good management is more important than good technology**
   - The best technology will not compensate for poor management, and a good manager can produce great results even with minimum resources. Good management motivates people to do their best, but there are no universal "right" styles of management.

The Principles of Conventional Software Engineering

13. **People are the key to success**
    - Highly skilled people with appropriate experience, talent, and training are key. The right people with insufficient tools, languages, and process will succeed. The wrong people with appropriate tools, languages, and process will probably fail.
14. **Follow with care**
    - Everybody is doing something but does not make it right for you. It may be right, but you must carefully assess its applicability to your environment. Object orientation, measurement, reuse, process improvement, CASE, prototyping-all these might increase quality, decrease cost, and increase user satisfaction. The potential of such techniques is often oversold, and benefits are by no means guaranteed or universal.
15. **Take responsibility**
    - When a bridge collapses we ask “What did the engineer do wrong?” Similarly if the software fails, we ask the same. So the fact is in every engineering discipline, the best methods can be used to produce poor results and the most out of date methods to produce stylish design.

The Principles of Conventional Software Engineering

16. **Understand the customer's priorities**
    - It is possible the customer would tolerate 90% of the functionality delivered late if they could have 10% of it on time.
17. **The more they see, the more they need**
    - The more functionality (or performance) you provide a user, the more functionality (or performance) the user wants.
18. **Plan to throw one away**
    - One of the most important critical success factors is whether or not a product is entirely new. Such brand-new applications, architectures, interfaces, or algorithms rarely work the first time.
19. **Design for change**
    - The architectures, components, and specification techniques you use must accommodate change.

The Principles of Conventional Software Engineering

20. Design without documentation is not design

- I have often heard software engineers say, “I have finished the design. All that is left is the documentation.”

21. Use tools, but be realistic

- Software tools make their users more efficient.

22. Avoid tricks

- Many programmers love to create programs with tricks-constructs that perform a function correctly, but in an obscure way. Show the world how smart you are by avoiding tricky code.

23. Encapsulate

- Information-hiding is a simple, proven concept that results in software that is easier to test and much easier to maintain.

The Principles of Conventional Software Engineering

24. Use coupling and cohesion

- Coupling and cohesion are the best ways to measure software's inherent maintainability and adaptability.

25. Use the McCabe complexity measure

- Although there are many metrics available to report the inherent complexity of software, none is as intuitive and easy to use as Tom McCabe's.

26. Don't test your own software

- Software developers should never be the primary testers of their own software.

The Principles of Conventional Software Engineering

27. Analyze causes for errors

- It is far more cost-effective to reduce the effect of an error by preventing it than it is to find and fix it. One way to do this is to analyze the causes of errors as they are detected.

28. Realize that software's entropy increases

- Any software system that undergoes continuous change will grow in complexity and become more and more disorganized.

29. People and time are not interchangeable

- Measuring a project solely by person-months makes little sense.

30. Expert excellence

- Your employees will do much better if you have high expectations for them.

The Principles of Modern Software Management

- Although the current software management principles described in previous Section evolved from and improved on conventional techniques, they still do not emphasize the modern principles.
- Building on Davis's format, here are top 10 principles of modern software management.

1. **Base the process on an architecture-first approach**
   - This requires that a demonstrable balance be achieved among the driving requirements, design decisions, and the life-cycle plans before the resources are committed for full-scale development.

The Principles of Modern Software Management

2. **Establish an iterative life-cycle process that confronts risk early.**
   - With today's sophisticated software systems, it is not possible to define the entire problem, design the entire solution, build the software, then test the end product in sequence. Instead, an iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages a balanced treatment of all stakeholder objectives.
   - Major risks must be addressed early to increase predictability and avoid expensive downstream scrap and rework.

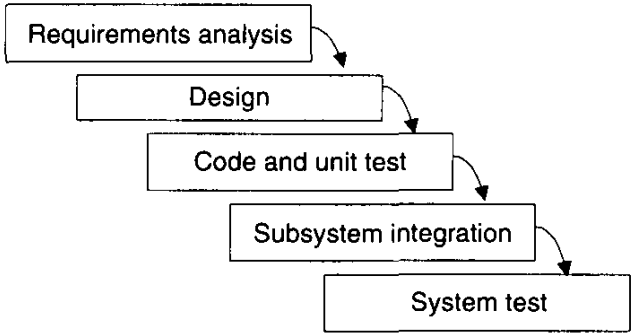
The Principles of Modern Software Management

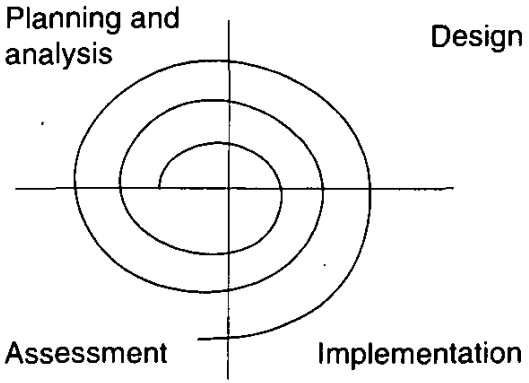
3. **Transition design methods to emphasize component-based development.**
   - Moving from a line-of-code mentality to a component-based mentality is necessary to reduce the amount of human-generated source code and custom development.
   - A **component** is a cohesive set of preexisting lines of code, either in source or executable format, with a defined interface and behavior.
4. **Establish a change management environment.**
   - The dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baselines.

The Principles of Modern Software Management

5. **Enhance change freedom through tools that support round-trip engineering.**
   - Round-trip engineering is the environment support necessary to automate and synchronize engineering information in different formats (such as requirements specifications, design models, source code, executable code, test cases).
   - Change freedom is a necessity in an iterative process, and establishing an integrated environment is crucial.
6. **Capture design artifacts in rigorous, model-based notation.**
   - A model-based approach (such as UML) supports the evolution of semantically rich graphical and textual design notations.
   - Visual modeling with rigorous notations and a formal machine-processable language provides more objective measures than the traditional approach of human review and inspection of ad hoc design representations in paper documents.

Waterfall Process	Iterative Process
Requirements first	***Architecture first***
Custom development	***Component-based development***
Change avoidance	***Change management***
Ad hoc tools	***Round-trip engineering***





Architecture-first approach → **The central design element**

Design and integration first, then production and test

Iterative life-cycle process → **The risk management element**

Risk control through ever-increasing function, performance, quality

Component-based development → **The technology element**

Object-oriented methods, rigorous notations, visual modeling

Change management environment → **The control element**

Metrics, trends, process instrumentation

Round-trip engineering → **The automation element**

Complementary tools, integrated environments

The Principles of Modern Software Management

7. **Instrument the process for objective quality control and progress assessment.**
   - Life-cycle assessment of the progress and the quality of all intermediate products must be integrated into the process.
   - The best assessment mechanisms are well-defined measures derived directly from the evolving engineering artifacts and integrated into all activities and teams.
8. **Use a demonstration-based approach to assess intermediate artifacts.**
   - Transitioning the current state-of-the-product artifacts (whether the artifact is an early prototype, a baseline architecture, or a beta capability) into an executable demonstration of relevant scenarios stimulates earlier convergence on integration, a more tangible understanding of design trade-offs, arid earlier elimination of architectural defects.

The Principles of Modern Software Management

9. **Plan intermediate releases in groups of usage scenarios with evolving levels of detail.**
   - It is essential that the software management process drive toward early and continuous demonstrations within the operational context of the system, namely its use cases.
   - The evolution of project increments and generations must be commensurate with the current level of understanding of the requirements and architecture.
   - Cohesive usage scenarios are then the primary mechanism for organizing requirements, defining iteration content, assessing implementations, and organizing acceptance testing.
10. **Establish a configurable process that is economically scalable.**
   - No single process is suitable for all software developments.
   - A pragmatic process framework must be configurable to a broad spectrum of applications. The process must ensure that there is economy of scale and return on investment by exploiting a common process spirit, extensive process automation, and common architecture patterns and components.

TABLE 4-1. *Modern process approaches for solving conventional problems*

CONVENTIONAL PROCESS: TOP 10 RISKS	IMPACT	MODERN PROCESS: INHERENT RISK RESOLUTION FEATURES
1. Late breakage and excessive scrap/rework	Quality, cost, schedule	Architecture-first approach Iterative development Automated change management Risk-confronting process
2. Attrition of key personnel	Quality, cost, schedule	Successful, early iterations Trustworthy management and planning
3. Inadequate development resources	Cost, schedule	Environments as first-class artifacts of the process Industrial-strength, integrated environments Model-based engineering artifacts Round-trip engineering
4. Adversarial stakeholders	Cost, schedule	Demonstration-based review Use-case-oriented requirements/testing
5. Necessary technology insertion	Cost, schedule	Architecture-first approach Component-based development
6. Requirements creep	Cost, schedule	Iterative development Use case modeling Demonstration-based review
7. Analysis paralysis	Schedule	Demonstration-based review Use-case-oriented requirements/testing
8. Inadequate performance	Quality	Demonstration-based performance assessment Early architecture performance feedback
9. Overemphasis on artifacts	Schedule	Demonstration-based assessment Objective quality control
10. Inadequate function	Quality	Iterative development Early prototypes, incremental releases

Transitioning to an Iterative Process

- Modern software development processes have moved away from the conventional waterfall model, in which each stage of the development process is dependent on completion of the previous stage.
- Development then proceeds as a series of iterations, building on the core architecture until the desired levels of functionality, performance, and robustness are achieved.
- An iterative process emphasizes the whole system rather than the individual parts.
- Risk is reduced early in the life cycle through continuous integration and refinement of requirements, architecture, and plans. The downstream surprises that have plagued conventional software projects are avoided.

Transitioning to an Iterative Process

- The economic benefits inherent in transitioning from the conventional waterfall model to an iterative development process are significant but difficult to quantify.
- As one benchmark of the expected economic impact of process improvement, consider the process exponent parameters of the COCOMO II model.
- This exponent can range from 1.01 (virtually no diseconomy of scale) to 1.26 (significant diseconomy of scale).
- The parameters that govern the value of the process exponent are application precedentedness, process flexibility, architecture risk resolution, team cohesion, and software process maturity.

Transitioning to an Iterative Process

- The following attributes map the process exponent parameters of COCOMO II to top 10 principles of a modern process.

1. **Application precedentedness**
   - Domain experience is a critical factor in understanding how to plan and execute a software development project.
   - For unprecedented systems, one of the key goals is to confront risks and establish early precedents, even if they are incomplete or experimental.
   - This is one of the primary reasons that the software industry has moved to an iterative life-cycle process.
   - Early iterations in the life cycle establish precedents from which the product, the process, and the plans can be elaborated in evolving levels of detail.

Transitioning to an Iterative Process

2. Process flexibility

- Development of modern software is characterized by such a broad solution space and so many interrelated concerns that there is a paramount need for continuous incorporation of changes.
- These changes may be inherent in the problem understanding, the solution space, or the plans.
- Project artifacts must be supported by an efficient change management environment commensurate with project needs.
- Both a rigid process and a chaotically changing process are destined for failure except with the most trivial projects.
- A configurable process that allows a common framework to be adapted across a range of projects is necessary to achieve a software return on investment.

Transitioning to an Iterative Process

3. Architecture risk resolution

- Architecture-first development is a crucial theme underlying a successful iterative development process.
- A project team develops and stabilizes an architecture before developing all the components that make up the entire suite of applications components.
- An architecture-first and component-based development approach forces the infrastructure, common mechanisms, and control mechanisms to be elaborated early in the life cycle and drives all component make/buy decisions into the architecture process.
- This approach initiates integration activity early in the life cycle as the verification activity of the design process and products.

Transitioning to an Iterative Process

4. Team cohesion

- Successful teams are cohesive, and cohesive teams are successful. It is not sure which is the cause and which is the effect, but successful teams and cohesive teams share common objectives and priorities.
- Cohesive teams avoid sources of project turbulence and entropy that may result from difficulties in synchronizing project stakeholder expectations.
- While there are many reasons for such turbulence, one of the primary reasons is miscommunication, particularly in exchanging information solely through paper documents that present engineering information subjectively.
- Advances in technology (such as programming languages, UML, and visual modeling) have enabled more rigorous and understandable notations for communicating software engineering information, particularly in the requirements and design artifacts that previously were ad hoc and based completely on paper exchange.
- These model-based formats have also enabled the round-trip engineering support needed to establish change freedom sufficient for evolving design representations.

Transitioning to an Iterative Process

5. Software process maturity

- The Software Engineering Institute's Capability Maturity Model (CMM) is a well-accepted benchmark for software process assessment.
- Software process maturity is crucial for avoiding software development risks and exploiting the organization's software assets and lessons learned.
- One of the key themes is that truly mature processes are enabled through an integrated environment that provides the appropriate level of automation to instrument the process for objective quality control.